Revisiting "Over-smoothing" in Deep GCNs

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, Tarek Abdelzaher
University of Illinois, Urbana-Champaign, IL 61801, USA
{chaoqiy2,syao9,ruijiew2,sl29,zaher}@illinois.edu

Abstract

Oversmoothing has been assumed to be the major cause of performance drop in deep graph convolutional networks (GCNs). The evidence is usually derived from Simple Graph Convolution (SGC), a linear variant of GCNs. In this paper, we revisit graph node classification from an optimization perspective and argue that GCNs can actually learn anti-oversmoothing, whereas *overfitting* is the real obstacle in deep GCNs. This work interprets GCNs and SGCs as two-step optimization problems and provides the reason why deep SGC suffers from oversmoothing but deep GCNs does not. Our conclusion is compatible with the previous understanding of SGC, but we clarify why the same reasoning does not apply to GCNs. Based on our formulation, we provide more insights into the convolution operator and further propose a *mean-subtraction* trick to accelerate the training of deep GCNs. We verify our theory and propositions on three graph benchmarks. The experiments show that (i) in GCN, overfitting leads to the performance drop and oversmoothing does not exist even model goes to very deep (100 layers); (ii) *mean-subtraction* speeds up the model convergence as well as retains the same expressive power; (iii) the weight of neighbor averaging (1 is the common setting) does not significantly affect the model performance once it is above the threshold (> 0.5).

1 Introduction

Graph neural networks (GNNs) are widely used in modeling real-world connections, like protein networks [1], social networks [2], and co-author networks [3]. One could also construct similarity graphs by linking data points that are close in the feature space even when there is no explicit graph structure. There have been several successful GNN architectures: ChebyshevNet [4], GCN [3], SGC [5], GAT [6], GraphSAGE [7] and other subsequent variants tailored for practical applications [8, 9, 10].

Recently, researchers started to explore the fundamentals of GNNs, such as expressive power [11, 12, 13, 14], and analyze their capacity and limitations. One of the frequently mentioned limitations is *oversmoothing* [15]. In deep GCNs, *over-smoothing* means that after multi-layer graph convolution, the effect of Laplacian smoothing makes node representations become more and more similar, eventually becoming indistinguishable. This issue was first mentioned in [16] and has been widely discussed since then, such as in JKNet [17], DenseGCN [15], DropEdge [18], and PairNorm [19]. However, these discussions were mostly on the powering effect of convolution operator A^L (where A is the convolution operator, and L is the number of layers). This essentially implies a GCN variant without an activation function (i.e., a simplified graph convolution, or SGC [5]).

In this work, we instead argue that *deep GCNs can learn anti-oversmoothing, while overfitting is the real cause of performance drop.* This paper focuses on graph node classification and starts from the perspective of graph-based optimization (minimizing $\mathcal{L}_0 + \gamma\mathcal{L}_{reg}$) [20, 21], where $\mathcal{L}_0$ is the (supervised) empirical loss and $\mathcal{L}_{reg}$ is an (unsupervised) graph regularizer, which encodes smoothness over the connected node pairs. We interpret a GCN as a two-step optimization problem: (i) forward propagation to minimize $\mathcal{L}_{reg}$ by viewing $\{W^{(l)}\}$ as constants and $\{X^{(l)}\}$ as parameters,

Preprint. Under review.

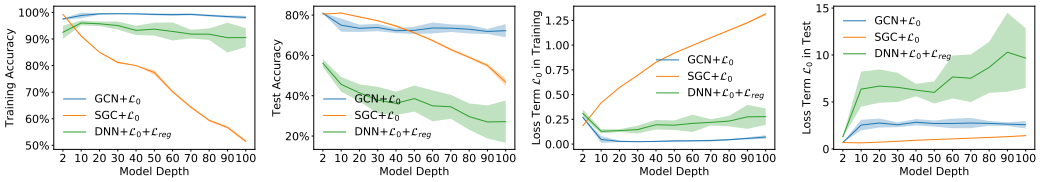

Figure 1: Comparison of Deep GCN, Deep SGC and DNN on *Cora*

and (ii) back propagation to minimize $\mathcal{L}_0$ by updating $\{W^{(l)}\}$. This work therefore derives GCN as:

$$\text{STEP1: } \min_{\{X^{(l)}\}} \mathcal{L}_{reg}\left(\{X^{(l)}\} \mid \{W^{(l)}\}\right) \quad \text{and} \quad \text{STEP2: } \min_{\{W^{(l)}\}} \mathcal{L}_0\left(\{W^{(l)}\}\right)$$

Similarly, an SGC is interpreted as:

$$\text{STEP1: } \min_{X} \mathcal{L}_{reg}(X) \quad \text{and} \quad \text{STEP2: } \min_{W} \mathcal{L}_0(W)$$

From this formulation, we show that a deep SGC indeed suffers from oversmoothing but deep GCNs will learn to prevent oversmoothing because (i) $\mathcal{L}_{reg}$ is conditioned on $\{W^{(l)}\}$ in GCN, and (ii) $\mathcal{L}_0$ and $\mathcal{L}_{reg}$ are somehow contradicted. To prevent gradient vanishing/exploding, in this paper, we add skip connections [22] to all deep architectures by default. An illustration of deep GCNs, deep SGC and directly learning $\mathcal{L}_0 + \gamma\mathcal{L}_{reg}$ using DNN is shown in Fig. 1.

As is mentioned above, the training of deep GCNs is a learning process of anti-oversmoothing, which is extremely slow in practice and sometimes may not converge. Based on the formulation, we further propose a *mean-subtraction* trick to accelerate the training of deep GCNs. Extensive experiments verify our theories and provide more insights about deep GCNs.

2 Background of Graph Transductive Learning

Graph representation learning aims at embedding the nodes into low-dimensional vectors, while simultaneously preserving both *graph topology structure* and *node feature information*. Given a graph $G = (V, E)$, let $V = \{v_1, v_2, \cdots, v_n\}$ be the set of nodes, and let Y be a set of m possible classes. Assume that each node v_j is associated with a class label $y_j \in Y$. A graph could be represented by an *adjacency matrix* A with $A_{ij} = 1$ when two nodes are connected $(v_i, v_j) \in E$. The *degree matrix* $D = diag(d_1, d_2, \ldots, d_n)$ is diagonal where $d_i = \sum_j A_{ij}$. Let $X = \{x_1, x_2, \ldots, x_n\}$ denote the feature vectors for each node. Given a labelled set $T \subset V$, the goal of transductive learning on a graph is to transductively predict labels for the remaining unknown nodes $V \setminus T$. A well-studied solution category is to include graph regularizers [23, 20, 24, 25] into the classification algorithm. Graph-convolution-based models [3, 4, 6, 7] are a powerful learning approach in this space.

2.1 Graph-based Regularization

There is a rather general class of embedding algorithms that include graph regularizers. They could be described as: finding a mapping $f(\cdot)$ by minimizing the following two-fold loss:

$$\mathcal{L} = \mathcal{L}_0\left(f(X)\right) + \gamma\mathcal{L}_{reg}\left(f(X)\right), \tag{1}$$

where $f(X) = [f(x_i)]_{i=1}^n$ is the low-dimensional representation of nodes. The first term is the empirical risk on the labelled set T. The second term is a graph regularizer over connected pairs, so as to make sure that a trivial solution is not reached.

The measurements on graphs are usually invariant to node permutations. A canonical way is to use Dirichlet energy [26] for the graph-base regularization,

$$\mathcal{L}_{reg} = \frac{1}{2}\sum_{i,j} A_{ij}\left\|\frac{f(x_i)}{\sqrt{d_i}} - \frac{f(x_j)}{\sqrt{d_j}}\right\|^2 = \frac{1}{2}\,\mathrm{Tr}\left(f(X)^\top \Delta f(X)\right), \tag{2}$$

where $\Delta = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the *normalized Laplacian operator*, which induces a semi-norm on $f(\cdot)$, penalizing the changes between adjacent vertices. Same normalized formulation could be found in [27, 28, 29, 30, 20], and some related literature also use the unnormalized version [24, 31].

2.2 Graph Convolutional Network

GCNs are derived from graph signal processing [32, 33, 34]. On the spectral domain, the operator Δ is a real-valued symmetric semidefinite matrix and the graph convolution is parameterized by a learnable filter g_θ on the its eigenvalue matrix. Kipf et al. [3] made assumptions of the largest eigenvalue (i.e., $\lambda_{max} = 2$) and simplified it with two-order Chebyshev expansion,

$$g_\theta \star x \approx \theta_0 x + \theta_1(\Delta - I)x \approx \theta(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x. \tag{3}$$

A multi-layer graph convolutional network (GCN) is formulated as the following layer-wise propagation rule ($\sigma(\cdot)$ is an activation function, e.g., ReLU):

$$X^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}(I + A)\tilde{D}^{-\frac{1}{2}}X^{(l)}W^{(l)}\right) \tag{4}$$

where $\tilde{D}^{-\frac{1}{2}}(I + A)\tilde{D}^{-\frac{1}{2}} \leftarrow I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the renormalization trick, $X^{(l)}$ and $W^{(l)}$ are the layer-wise feature and parameter matrices, respectively.

3 GCN as Two-Step Optimization

In this section, we re-interpret a GCN as a two-step optimization problem, where STEP1 is to minimize $\mathcal{L}_{reg}$ by viewing $\{W^{(l)}\}$ as constants and $\{X^{(l)}\}$ as parameters while STEP2 minimizes $\mathcal{L}_0$ by updating $\{W^{(l)}\}$. Overall, the GCN architecture is interpreted as a layer-wise combination of MLP architecture and the gradient descent algorithm of minimizing $\mathcal{L}_{reg}$. In the meantime, the training process of parameters is entirely inherited from MLP, which aims at only minimizing $\mathcal{L}_0$. Let us first discuss a gradient descent algorithm for minimizing the form $\frac{1}{2}\,\mathrm{Tr}(X^\top \Delta X)$.

3.1 Gradient Descent for Trace Optimization.

Problem Definition. Given the Laplacian operator $\Delta \in \mathbb{R}^{n\times n}$, we consider to minimize the trace on feature domain $X \in \mathbb{R}^{n\times m}$, where m is the input dimension. To prevent the trivial solution $X = \mathbf{0} \in \mathbb{R}^{n\times m}$, we consider the energy constraint on X, i.e., $\|X\|_F^2 = c_1 \in \mathbb{R}^+$. The trace optimization problem is:

$$\min \ \frac{1}{2}\,\mathrm{Tr}(X^\top \Delta X),\ \textit{subject to const.}\ \|X\|_F^2, \tag{5}$$

where $\|X\|_F^2$ denotes the Forbenius-norm of X. To solve this, We equivalently transform the optimization problem into the *Reyleigh Quotient* form $R(X)$, which is,

$$\min \ R(X) = \frac{\frac{1}{2}\,\mathrm{Tr}(X^\top \Delta X)}{\|X\|_F^2} = \frac{\frac{1}{2}\,\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)},\ \textit{subject to const.}\ \|X\|_F^2. \tag{6}$$

It is obvious that $R(X)$ is scaling invariant on X, i.e., $\forall\, c_2 \neq 0 \in \mathbb{R}$, $R(X) = R(c_2 \cdot X)$.

One-step Improvement. Suppose an initial guess is X, one-step of trace optimization aims at finding a better guess X'', which satisfies $R(X'') \leq R(X)$ and $\|X''\|_F^2 = c_1$. Our strategy is first view the problem as unconstrainted optimization on $R(X)$ and update the guess X to X' by gradient descent. Then we rescale X' and reach the improved guess X'', which meets the norm constraint.

Given the initial guess X, we move against the derivative of $R(X)$ by the learning rate $\eta = \mathrm{Tr}(X^\top X)/(2 - \frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)})$ and reach an intermediate solution X' in the unconstrainted space:

$$\nabla_X = \frac{\partial R(X)}{\partial X} = \frac{1}{2}\frac{\partial \frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)}}{\partial X} = \frac{\left(\Delta - I\frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)}\right)X}{\mathrm{Tr}(X^\top X)}, \tag{7}$$

$$X' = X - \eta\nabla_X = \frac{(2-\Delta)X}{2 - \frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)}} = \frac{(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})}{2 - \frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)}}X. \tag{8}$$

Immediately, we get $R(X') \leq R(X)$ (details in Appendix B). Then, we rescale X' (to achieve the improved guess $X'' = c_2 \cdot X'$, which naturally satisfies $R(X'') = R(X')$) by a constant $c_2 \in \mathbb{R}^+$

to meet the norm constraint, such that $\|X''\|_F^2 = c_1$. Therefore, the improved guess satisfies $R(X'') = R(X') \leq R(X)$ and has the following form,

$$X'' = c_2 \cdot X' \propto (I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X. \tag{9}$$

Note that, if we conduct the trace optimization algorithm enough times, the optimal solution will finally be proportional to the largest eigenvector of Δ, which causes oversmoothing.

3.2 Layer-wise Propagation and Optimization

We introduce the trace optimization solution into the layer-wise propagation of MLP. Given the node set V, features $X = \{x_1, x_2, \ldots, x_n\}$ and a labelled set $T \subset V$, a label mapping, $f_{\{W^{(l)}\}} : X \mapsto Y$, is usually a deep neural network, which could be tailored according to the practical applications. For example, $f(\cdot)$ could be a convolutional neural network (CNN) for image recognition or a recurrent neural network (RNN) for language processing. In this scenario, we first consider a simple multi-layer perceptron (MLP). The forward propagation rule of an MLP is given by,

$$X^{(l+1)} = \sigma(X^{(l)}W^{(l)}), \tag{10}$$

where $W^{(l)}$ and $X^{(l)}$ are layer-wise parameters and inputs.

STEP1: minimizing $\mathcal{L}_{reg}$ in Forward Propagation. Let us fix parameters $\{W^{(l)}\}$ and consider $X^{(l)}$, i.e., the output of $(l-1)_{th}$ layer, as an initial guess of the trace optimization problem. We know from Sec. 3.1 that through one-step gradient descent, Eqn. (9) will find an improved guess for minimizing $\mathcal{L}_{reg}$:

$$X^{(l)''} = c_2 \cdot X^{(l)'} = X^{(l)} - \eta \nabla_X \Big|_{X=X^{(l)}} \propto (I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X^{(l)}. \tag{11}$$

We plug this new value into Eqn. (10) and immediately reach the same convolutional propagation rule, $f^{(l)} : X^{(l)} \mapsto X^{(l+1)}$, as Kipf et al. [3] (before applying the renormalization trick),

$$X^{(l+1)} = \sigma\left((I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X^{(l)}W^{(l)}\right), \tag{12}$$

where the constant scalar in Eqn. (11) is absorbed into parameter matrix $W^{(l)}$. Therefore, a GCN forward propagation is essentially applying STEP1 layerwise in the forward propagation of an MLP, which is formulated as a composition of mappings $f = f^{(L-1)} \circ \cdots \circ f^{(0)}$ on initial feature $X^{(0)}$.

STEP2: minimizing $\mathcal{L}_0$ in Back Propagation. After forward propagation, the cross-entropy loss $\mathcal{L}_0$ over the labelled set is calculated. In this procedure, we then conversely view $\{X^{(l)}\}$ as constants and $\{W^{(l)}\}$ as parameters of the MLP and conduct standard back-propagation algorithm.

3.3 GCN: combining STEP1 and STEP2

In essence, STEP1 essentially defines a combined architecture, where the layer-wise propagation is adjusted by an additional step of trace optimization. In STEP2, under that architecture, the optimal $\{W^{(l)}\}$ is learned and a low-dimension $f(X)$ is reached with respect to $\mathcal{L}_0$ explicitly and $\mathcal{L}_{reg}$ implicitly, after standard loss back-propagation. We express it as a two-step optimization,

$$\underbrace{\text{STEP1: } \min_{\{X^{(l)}\}} \mathcal{L}_{reg}\left(\{X^{(l)}\} \mid \{W^{(l)}\}\right)}_{why\ GCN\ architecture?} \quad \text{and} \quad \underbrace{\text{STEP2: } \min_{\{W^{(l)}\}} \mathcal{L}_0\left(\{W^{(l)}\}\right)}_{train\ the\ architecture.}. \tag{13}$$

In this section, the learning rate $\eta = \text{Tr}(X^\top X)/(2 - \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)})$ is specially chosen, and it satisfies $\eta \in (0, \infty)$ since $X^\top X$ is semi-definite and $\frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)}$ is smaller than the largest eigenvalue of Δ, which is smaller than 2. In the experiment section, we reveal that η is related to the weight of neighbor information averaging. We further test different η and provide more insights on graph convolution operators. In the following sections, we use A_{sym} to denote the convolutional operator $\tilde{D}^{-\frac{1}{2}}(I + A)\tilde{D}^{-\frac{1}{2}}$ and use A_{rw} for the random walk form $\tilde{D}^{-1}(I + A)$.

4 The Over-smoothing Issue

The recent successes in applying GNNs are largely limited to *shallow* architectures (e.g., 2-4 layers). Model performance decreases when adding more intermediate layers. Summarized in [19], there are three possible contributing factors: (i) overfitting due to increasing number of parameters (one matrix $W^{(l)}$ per layer); (ii) gradient vanishing/exploding; (iii) oversmoothing due to Laplacian smoothing. The first two points are common to all deep neural networks. The issue of oversmoothing is therefore the focus in this work. We show that *deep GCNs can learn anti-oversmoothing by nature, but overfitting is the major cause of performance drop*.

In this section, we first recall the commonly discussed oversmoothing problem (in SGC). Based on the re-formulation in Sec. 3.3, we then show how deep GCNs have the ability to learn anti-oversmoothing. Further, we propose an easy but effective *mean-subtraction* trick to speed up anti-oversmoothing, which accelerates the convergence in training deep GCNs.

4.1 Over-smoothing in SGC.

Oversmoothing means that node representations become similar and finally go indistinguishable after multi-layer graph convolution. Starting from the random walk theory, the analysis for oversmoothing is usually done on a connected, un-directed and non-bipartite graph. The issue is discussed in [16, 17, 15, 18, 19] and mainly on the L-layer linear SGC [5].

SGC was proposed in [5], with the hypothesis that the non-linear activation is not critical while the majority of the benefit arises from the local averaging A. The authors directly remove the activation function and proposed a linear "L-layer" model,

$$f(X) = A\left(A(\cdots)W^{(L-1)}\right)W^{(L)} = A^L XW. \tag{14}$$

where $\prod W^{(l)}$ has collapsed into a single W. This model explicitly disentangles the dependence of STEP1 and STEP2. We similarly formulate the SGC model in the form of two-step optimization,

$$\text{STEP1: } \min_X \mathcal{L}_{reg}(X) \quad \text{and} \quad \text{STEP2: } \min_W \mathcal{L}_0(W) \tag{15}$$

Theorem 1. *Given any random signal $x \in \mathbb{R}^n$ and a symmetric matrix $A \in \mathbb{R}^{n\times n}$, the following property $\lim_{k\to\infty} A^k x \propto u_1$ holds almost everywhere on x, where A has non-negative eigenvalues and u_1 is the eigenvector associated with the largest eigenvalue of A.*

For two widely used convolution operators A_{sym} and A_{rw}, they have the same dominant eigenvalue $\lambda_1 = 1$ with eigenvectors $\tilde{D}^{\frac{1}{2}}\mathbf{1}$ and $\mathbf{1}$, respectively. From the two-step optimization form, SGC is essentially conducting gradient descent algorithm L times in STEP1. According to Theorem 1 (see proofs in Appendix D), if L goes to infinity, then each output feature channel will become $\lim_{k\to\infty} A^k_{sym}x \propto \tilde{D}^{\frac{1}{2}}\mathbf{1} \in \mathbb{R}^n$ or $\lim_{k\to\infty} A^k_{rw}x \propto \mathbf{1} \in \mathrm{R}^n$, which means oversmoothing. In STEP2, SGC model will seek to minimize L_0 on the basis of the oversmoothed features. The independence between STEP1 and STEP2 accounts for the performance drop in deep SGC.

4.2 Anti-oversmoothing in GCN.

On the contrary, the result of $\mathcal{L}_{reg}$ in STEP1 is dependent on $\{W^{(l)}\}$, i.e., also on STEP2, in GCN. In fact, after STEP1, node representation in GCN will be oversmoothed to some extent as well. However, during STEP2, GCN will learn to update layer-wise $\{W^{(l)}\}$ and make node features separable, such that $\mathcal{L}_0$ will be minimized, during which, the effect of STEP1 (minimizing $\mathcal{L}_{reg}$, i.e., making node features inseparable) will be mitigated and $\mathcal{L}_{reg}$ actually increases implicitly. In essence, the dependency of $\mathcal{L}_{reg}$ enables GCNs to do anti-ovesmoothing during STEP2.

We demonstrate on the *Karate* club dataset: this graph has 34 vertices of 4 classes (the same labeling as [3, 35]) and 78 edges. A 32-layer GCN (deep enough for this demo dataset), with 16 hidden units in each layer, is considered. The model is trained on identity feature matrix with basic residual connection [22]. The training set T consists of two labeled examples per class. After 1000 epochs, the model achieves 96.15% accuracy in the testing samples. We present the feature-wise smoothing by layer in Fig. 2.a and node-wise smoothing by layer in Fig. 2.b. The y-axis score of the first two

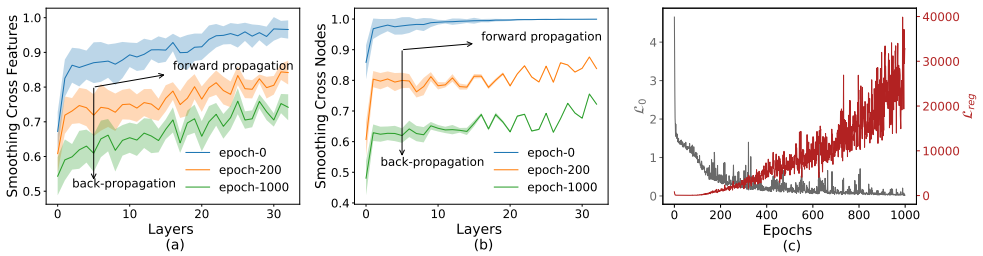

Figure 2: Feature-wise and Node-wise Smoothing during GCN Training on *Karate*

figures are calculated by cosine similarity. We also calculate and present $\mathcal{L}_0$ as well as $\mathcal{L}_{reg}$ for each training epoch in Fig. 2.c. More details of the demo are given in Appendix F.

From the demonstrations, we observe that without training (blue curves in Fig. 2.a and Fig. 2.b and the beginning of Fig. 2.c), the issue of oversmoothing do exist for deep GCNs. Because forward propagation mixes up features layer-by-layer. However, this issue is automatically addressed during training and with more training epochs, feature-wise smoothing and node-wise smoothing are gradually mitigated. The effect is more obvious when referring to Fig. 2.c, where $\mathcal{L}_{reg}$ actually increases during the training (small $\mathcal{L}_{reg}$ indicates oversmoothing). The gradual increase in $\mathcal{L}_{reg}$ demonstrates that GCNs have the ability to learn to *anti-oversmooth* by nature. Then, the next question comes: what is the real cause of performance drop in deep GCNs? Our answer is *overfitting*. We give practical support in the experimental section.

4.3 Mean-subtraction: an Accelerator

Although deep GCNs could learn anti-oversmoothing naturally, another practical problem appears that the convergence of training deep GCNs is extremely slow (sometimes may not converge). This issue has not been explored extensively in the literature. In this work, we present a *mean-subtraction* trick to accelerate the training of deep GCNs, which theoretically magnifies the effect of Fiedler vector. PairNorm [19] also includes a *mean-subtraction* step, however, their purpose is to simplify derivation. This section provides more insights and motivation to use *mean-subtraction*.

There are primarily two reasons to use mean-subtraction: (i) deep neural network classifiers are discriminators that draw a boundary between classes. Therefore, the mean feature of the entire dataset (a DC signal) does not help with the classification, whereas the components away from the center (the AC signal) matters; (ii) layer-wise *mean-subtraction* will eliminate the dominant eigen component ($\tilde{D}^{\frac{1}{2}}\mathbf{1}$ or $\mathbf{1}$) and actually magnifies the Fiedler vector (the eigenvector associated with the second smallest eigenvalue of Δ), which reveals important community information and graph conductance [36]. This helps to set an initial graph partition and spends up model training (STEP2).

We start with one of the most popular convolution operator A_{rw} and its largest eigenvector $u_1 = \mathbf{1} \in \mathbb{R}^n$. Given any non-zero $x \in \mathbb{R}^n$, the *mean-subtraction* gives,

$$x_{new} \leftarrow x - x_{mean} = x - \frac{\mathbf{1}\mathbf{1}^\top x}{n} = x - \langle x, \bar{u_1} \rangle \cdot \bar{u_1} \tag{16}$$

where $\bar{u_1} = \frac{u_1}{\|u_1\|}$. Eqn. (16) reveals that *mean-subtraction* reduces the components aligned with $\{u_1\}$-space. This is exactly a step of numerical approximation of the Fiedler vector, which sets the initial graph partition (demonstration in Appendix F) and makes the feature separable. For A_{sys}, the formulation could be adjusted by a factor of $\tilde{D}^{\frac{1}{2}}$ (refer to Appendix E).

5 Experiments

In this section, we present experimental evidence to answer the following three questions: (i) whether oversmoothing is an issue in deep GCNs and why? (ii) How to stabilize and accelerate the training of deep GCNs? (iii) Does the learning rate η matter? How about changing them? We also provide more insights and draw useful conclusions for the practical usage of GCN models and its variants.

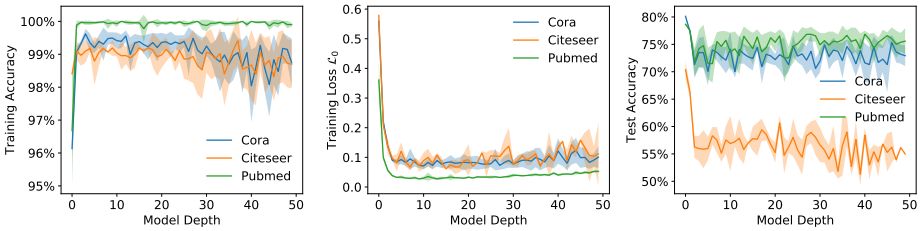

Figure 3: Vanilla Deep GCNs on Three Citation Networks

The experiments show the performance of deep GCNs on the semi-supervised node classification tasks. All the deep models (with more than 3 hidden layers) are implemented with basic skip-connection [3, 22]. Since skip-connection (also called residual connection) are necessary in deep architectures, we do not consider them as new models. Three benchmark citation networks (*Cora*, *Citeseer*, *Pubmed*) are considered. We follow the same experimental settings from [21] and show the basic statistics of datasets in Table. 1. All the experiments are conducted 20 times and mainly finished in a Linux server with 64GB memory, 32 CPUs and a single GTX-2080 GPU.

Dataset	Nodes	Edges	Features	Class	Label rate
Cora	2,708	5,429	1,433	7	0.052
Citeseer	3,327	4,732	4,732	6	0.036
Pubmed	19,717	44,338	500	3	0.003

Table 1: Overview of Citation Network Statistics

5.1 Overfitting in Deep GCNs

The performance of GCNs is known to decrease with increasing number of layers, for which, a common explanation is "oversmoothing". In Sec. 4, we contradict this thesis and conjecture instead that *overfitting is the major reason for the drop of performance in deep GCNs; we show that deep GCNs actually learn anti-oversmoothing*. In this section, we provide evidence to support our conjecture.

Performance vs Depth. We first evaluate vanilla GCN models (with residual connection) with $1 \sim 50$ hidden layers on *Cora, Citeseer* and *Pubmed*. The results of training and test accuracy are reported in Fig. 3.

Form Fig. 3, we know immediately that test accuracy drops in the beginning (1-4 layers) and then remains stable even as the model depth increases, which means the increasing number (≥ 4) of hidden layers does not hurt model performance. Thus, oversmoothing is not the reason. From 2 to 3 or 3 to 4 layers, we notice that these is a big rise in training accuracy (up to $\geq 99\%$) and simultaneously a big drop in training loss (to ≤ 0.1) and test accuracy consistently on the three datasets. This is more consistent with overfitting.

Deep GCNs Learn Anti-oversmoothing. We recall that in Sec. 4.2, we show from an optimization perspective that the dependency of $\mathcal{L}_{reg}$ on $\{W^{(l)}\}$ allows the network to learn anti-oversmoothing. To verify our theory, we compare SGC and GCN on *Cora* and *Pubmed* with various depth. To make it clear, SGC is actually a linear model, the depth L means the number of graph convolution S^L. Model performance in both training and test is shown in Fig. 4.

It is interesting that the accuracy of SGC decreases rapidly [12] with more graph convolutions either for training or test. This is a strong indicator of oversmoothing, because node features converge to the same stationary due to the effect of STEP1 (specified in Theorem 1). The performance of the GCN model is not as good as SGC soon after 2 layers because of overfitting, but it stabilizes at a high accuracy even as the model goes very deep, which again verifies that GCNs naturally have the power of anti-oversmoothing.

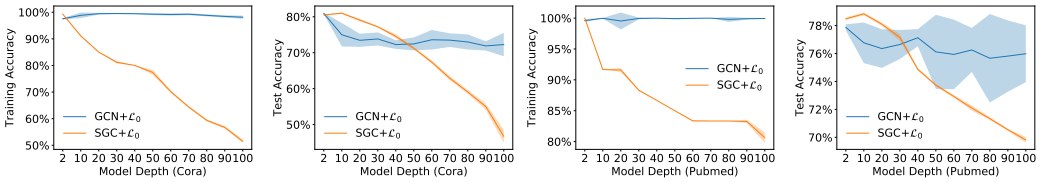

Figure 4: Comparison of Deep GCN and Deep SGC on *Cora* and *Pubmed*

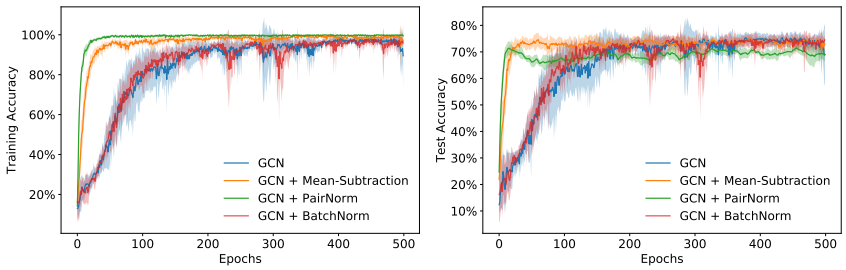

Figure 5: Comparison of Effective Tricks to Train Deep GCNs (20 trials)

5.2 Mean-Subtraction

To facilitate the training of deep GCNs, we proposed mean-subtraction in Sec. 4.3. In this section, we evaluate the efficacy of the mean-subtraction trick and compare it with vanilla GCNs [3], PairNorm [19] and the widely used BatchNorm [37] in the deep learning area. The four models have same settings, such as the number of layers (64), learning rate (0.01), and hidden units (16). Mean-subtraction is to subtract the mean feature value before each convolution layer (and PairNorm further re-scales the feature by variance). They do not include additional parameters. BatchNorm adds more parameters for each layer and learns to whiten the input of each layer. The experiment is conducted on *Cora* with 500 training epochs.

Fig. 5 reports the training and test curve for the four model variants. GCN and GCN+BatchNorm perform similarly, which means BatchNorm does not help substantially in training deep GCNs (at least for *Cora*). GCN+mean-subtraction and GCN+PairNorm give fast and stable training/test convergence. However, we could tell from the training curve that the PairNorm trick seems to suffer a lot from overfitting, leading to a drop in test accuracy. In sum, *mean-subtraction* not only speeds up the model convergence but also retains the same expressive power. It is an ideal trick for training deep GCNs.

5.3 Performance vs Learning Rate η

In Sec. 3, we choose the learning rate $\eta = \text{Tr}(X^\top X)/(2 - \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)})$. However, a different learning rate does lead to different weights $w(\eta)$ of neighbor information aggregation (we show that $w(\eta)$ is a monotonically increasing function in Appendix C). There are also some efforts on trying different ways to aggregate neighbor information [6, 7, 18, 38]. In this section, we consider the form "$I + w(\eta)A$" with $w(\eta) \in [0, \infty)$ and exploit a group of convolution operators by their normalized version. GCN with normalized $I + w(\eta)A$ is named as η-GCN. We evaluate this operator group on *Cora* and list the experimental results in Table. 2

Accuracy (%)		$w(\eta)$=**0**	**0.1**	**0.2**	**0.5**	**1.0**	**2**	**5**	**10**	**20**	**50**	**100**
2-layer	training	92.66	95.67	96.32	96.05	95.33	94.54	93.44	93.30	92.82	92.86	92.98
	test	52.66	71.62	75.64	78.26	79.72	79.06	78.34	78.14	77.93	77.67	77.58
32-layer	training	95.02	99.49	99.58	99.35	98.69	98.10	98.84	98.83	98.81	98.76	98.83
	test	39.93	72.53	73.59	73.65	74.03	75.11	74.16	75.08	75.49	74.64	74.74

Table 2: Performance vs Neighbor Averaging Weight (2-layer and 32-layer)

We conclude that when $w(\eta)$ is small (i.e., η is small), which means the gradient of $\mathcal{L}_{reg}$ does not contribute much to the end effect, η-GCN is more of a DNN. As $w(\eta)$ increases, a significant increase in model performance is initially observed. When $w(\eta)$ exceeds some threshold, the accuracy saturates, remaining high (or maybe decreases slightly in shallow models, i.e., 2-layer) even as we increase $w(\eta)$ substantially. We conclude that for the widely used shallow GCNs, the common choice of weight $w(\eta) = 1$, which means a learning rate, $\eta = \mathrm{Tr}(X^\top X)/(2 - \frac{\mathrm{Tr}(X^\top \Delta X)}{\mathrm{Tr}(X^\top X)})$, is large enough to include the gradient descent effect and small enough to avoid the drop in accuracy. To find the best weight of neighbor averaging, further inspection is needed in future work.

6 Conclusion

We reformulate GCNs from an optimization perspective by plugging the gradient of graph regularizer into a standard MLP. From this formulation, we revisit the commonly discussed "oversmoothing issue" in deep GCNs and provide a new understanding: *deep GCNs have the power to learn anti-oversmoothing by nature, but overfitting is the real cause of performance drop when the model goes deep*. We further propose a cheap but effective *mean-subtraction* trick to accelerate the training of deep GCNs. Extensive experiments are presented to verify our theory and provide more practical insights.

References

[1] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810, 2018.

[2] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[5] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[9] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems*, pages 8228–8239, 2019.

[10] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[11] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[12] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint cs.LG/1905.10947*, 2019.

[13] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

[14] Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems*, pages 15387–15397, 2019.

[15] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9267–9276, 2019.

[16] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[18] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

[19] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[20] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[21] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[24] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University, 2002.

[25] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural networks: Tricks of the trade*, pages 639–655. Springer, 2012.

[26] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[27] Yu Chen, Lingfei Wu, and Mohammed J Zaki. Deep iterative and adaptive learning for graph neural networks. *arXiv preprint arXiv:1912.07832*, 2019.

[28] Rie K Ando and Tong Zhang. Learning on graph with laplacian regularization. In *Advances in neural information processing systems*, pages 25–32, 2007.

[29] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.

[30] Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

[31] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[32] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

[33] Siheng Chen, Rohan Varma, Aliaksei Sandryhaila, and Jelena Kovačević. Discrete signal processing on graphs: Sampling theory. *IEEE transactions on signal processing*, 63(24):6510–6523, 2015.

[34] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[35] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[36] Zhengdao Chen, Xiang Li, and Joan Bruna. Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

[37] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[38] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

A $\mathcal{L}_{reg}$ and Spectral Clustering

Graph Regularizer $\mathcal{L}_{reg}$. $\mathcal{L}_{reg}$ is commonly formulated by Dirichlet energy, $\mathcal{L}_{reg} = \frac{1}{2}\,\mathrm{Tr}\left(H^\top \Delta H\right)$, where $f(\cdot)$ is a mapping from the input feature X to low-dimensional representation $H = f(X)$. To minimize $\mathcal{L}_{reg}$, this paper adds constraint on the magnitude of H, i.e., $\|H\|_F^2 = C \in \mathbb{R}$, which gives,

$$\min_H \ \frac{1}{2}\,\mathrm{Tr}\left(H^\top \Delta H\right),\ \textit{subject to const.}\ \|H\|_F^2. \tag{17}$$

Spectral Clustering. Given a graph with binary adjacency matrix A, a partition of node set V into k set could be written as $P_1, P_2, \ldots, P_k$ in graph theory. For normalized spectral clustering, the k indicator vectors is written as $h_i = (h_i^1, \ldots, h_i^n)$, where h_i^j represents the affiliation of node j in class set P_i and $vol(P_i) = \sum_{v_j \in P_i} d_j$ is the volume.

$$h_i^j = \begin{cases} \frac{1}{\sqrt{vol(P_i)}}, & v_j \ \in P_i \\ 0, & otherwise \end{cases} \tag{18}$$

The $H = [h_i^j]_{i=1..k, j=1..n}$ is a matrix containing these k indicator vectors as columns. For each row of H, there is only one non-empty entry, implying $h_i^\top h_j = 0,\ \forall i \neq j$. Let us revisit the *Normalized Cut* of a graph for a partition $P_1, P_2, \ldots, P_k$.

$$\begin{aligned}
\text{Ncut}(P_1, P_2, \ldots, P_k) &= \sum_{i=1}^{k} \frac{cut(P_i, \bar{P}_i)}{vol(P_i)} \\
&= \frac{1}{2}\sum_{i=1}^{k} \frac{\sum_{v_j \in P_i, v_t \notin P_i} A_{jt} + \sum_{v_j \notin P_i, v_t \in P_i} A_{jt}}{\sqrt{vol(P_i)}\sqrt{vol(P_i)}} \\
&= \frac{1}{2}\sum_{i=1}^{k} \left(\sum_{v_j \in P_i, v_t \notin P_i} \left(\frac{1}{\sqrt{vol(P_i)}}\right)^2 + \sum_{v_j \notin P_i, v_t \in P_i} \left(\frac{1}{\sqrt{vol(P_i)}}\right)^2 \right) \\
&= \frac{1}{2}\sum_{i=1}^{k}\sum_{j,t}(h_i^j - h_i^t)^2 = \frac{1}{2}\sum_{i=1}^{k} h_i^\top L h_i = \frac{1}{2}\mathrm{Tr}(H^\top L H).
\end{aligned} \tag{19}$$

Also, H satisfies $H^\top D H = I$. When the discreteness condition is relaxed and H is substitute by $H = D^{-\frac{1}{2}}U$, the normalized graph cut problem (normalized spectral clustering) is relaxed into,

$$\min_U \ \frac{1}{2}\,\mathrm{Tr}\left(U^\top \Delta U\right),\ \ \textit{subject to}\ \ U^\top U = I. \tag{20}$$

This is a standard trace minimization problem which is solved by the matrix the eigen matrix of Δ. Compared to Eqn. (17), Eqn. (20) has a stronger contraints, which outputs the optimal solution irrelevant to the inputs (feature matrix X). However, Eqn. (17) only add constraints on the magnitude of H, which balances the trade-off and will give a solution induced by both the eigen matrix of Δ and the original feature X.

B Reyleigh Quotient

Reyleigh Quotient. The Reyleigh Quotient of a vector $x \in \mathbb{R}^m$ is the scalar,

$$R(x) = \frac{x^\top \Delta x}{x^\top x}, \tag{21}$$

which is invariant to the scaling of x. For example, $\forall\ c_1 \neq 0 \in \mathbb{R}$, we have $R(x) = R(c_1 \cdot x)$. When we view $R(x)$ as a function on m-dim variable x, it has stationary points x_i, where x_i is the eigenvector of Δ. Let us assume $\Delta x_i = \lambda_i x_i$, then the stationary value at point x_i will be exactly the eigenvalue λ_i,

$$R(x_i) = \frac{x_i^\top \Delta x_i}{x_i^\top x_i} = \frac{x_i^\top \lambda_i x_i}{x_i^\top x_i} = \lambda_i. \tag{22}$$

When x is not an eigenvector of Δ, the partial derivatives of $R(x)$ with respect to the vector coordinate x_j is calculated as,

$$\begin{aligned}\nabla_{x_j} R(x) = \frac{\partial R(x)}{\partial x_j} &= \frac{\frac{\partial}{\partial x_j}(x^\top \Delta x)}{x^\top x} - \frac{(x^\top \Delta x)\frac{\partial}{\partial x_j}(x^\top x)}{(x^\top x)^2} \\ &= \frac{2(\Delta x)_j}{x^\top x} - \frac{(x^\top \Delta x)2x_j}{(x^\top x)^2} = \frac{2}{x^\top x}(\Delta x - R(x)x)_j \end{aligned} \tag{23}$$

Thus, the derivative of $R(x)$ with respect to x is collected as,

$$\nabla R(x) = \frac{2}{x^\top x}(\Delta x - R(x)x). \tag{24}$$

Minimizing R(x). Suppose $\Delta = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is the normalized Laplacian matrix. Let us first consider to minimize $R(x)$ without any constraints. Since Δ is a symmetric real-valued matrix, it could be factorized by Singular Value Decomposition,

$$\Delta = U \Lambda U^\top = \sum_{i=1}^{s} u_i \lambda_i u_i^\top \tag{25}$$

where s is the rank of Δ and $0 = \lambda_1 \leq \cdots \leq \lambda_s < 2$ are the eigen values. For any non-zero vector x, it is decomposed w.r.t. the eigen space of Δ,

$$x = \epsilon + \sum_{i=1}^{s} c_i \cdot u_i \tag{26}$$

where $\{c_i\}$ is the coordinates and ϵ is a component tangent to the eigen space spanned by $\{u_i\}$. Let us consider the component of x within the eigen space and discuss ϵ later. Therefore, the Reyleigh Quotient $R(x)$ can be calculated by,

$$R(x) = \frac{x^\top \Delta x}{x^\top x} = \frac{(\sum_{i=1}^{s} c_i \cdot u_i^\top)(\sum_{i=1}^{s} u_i \lambda_i u_i^\top)(\sum_{i=1}^{s} c_i \cdot u_i)}{(\sum_{i=1}^{s} c_i \cdot u_i^\top)(\sum_{i=1}^{s} c_i \cdot u_i)} = \frac{\sum_{i=1}^{s} c_i^2 \lambda_i}{\sum_{i=1}^{s} c_i^2} \tag{27}$$

Recall the partial derivative of $R(x)$ w.r.t. x in Eqn. (24). Think about to minimize $R(x)$ by gradient descent and always consider the learning rate (the same as what we used in the main text. The factor $\frac{1}{2}$ is from that the $R(x)$ in appendix does not have the scalar $\frac{1}{2}$),

$$\eta = \frac{1}{2}\text{Tr}(X^\top X)/(2 - \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)}) = \frac{1}{2} \cdot \frac{x^\top x}{2 - \frac{x^\top \Delta x}{x^\top x}} = \frac{1}{2} \cdot \frac{x^\top x}{2 - R(x)}. \tag{28}$$

The initial x is regarded as an starting point, and the next point x' is given by gradient descent,

$$x' = x - \eta \nabla R(x) = x - \frac{1}{2} \cdot \frac{x^\top x}{2 - R(x)} \frac{2}{x^\top x}(\Delta x - R(x)x) = \frac{2I - \Delta}{2 - R(x)} x. \tag{29}$$

The new Reyleigh Quotient value is,

$$R(x') = \frac{x'^\top \Delta x'}{x'^\top x'} = \frac{(\frac{2I-\Delta}{2-R(x)}x)^\top \Delta (\frac{2I-\Delta}{2-R(x)}x)}{(\frac{2I-\Delta}{2-R(x)}x)^\top (\frac{2I-\Delta}{2-R(x)}x)} = \frac{x^\top (2I-\Delta)\Delta(2I-\Delta)x}{x^\top (2I-\Delta)(2I-\Delta)x}. \tag{30}$$

The eigen properties of $2I - \Delta$ could be derived from Δ, where they have the same eigenvector, and any eigenvalue λ of Δ will adjust to be an eigenvalue $2 - \lambda$ of $2I - \Delta$. Therefore, we do further derivation,

$$R(x') = \frac{\sum_{i=1}^{s} c_i^2 (2 - \lambda_i)^2 \lambda_i}{\sum_{i=1}^{s} c_i^2 (2 - \lambda_i)^2}. \tag{31}$$

So far, to get the ideal effect, a final check is needed: whether the Reyleigh Quotient does decrease after the gradient descent.

$$\begin{aligned} R(x') - R(x) &= \frac{\sum_{i=1}^s c_i^2(2-\lambda_i)^2\lambda_i}{\sum_{i=1}^s c_i^2(2-\lambda_i)^2} - \frac{\sum_{i=1}^s c_i^2\lambda_i}{\sum_{i=1}^s c_i^2} \\ &= \frac{(\sum_{i=1}^s c_i^2)(\sum_{i=1}^s c_i^2(2-\lambda_i)^2\lambda_i) - (\sum_{i=1}^s c_i^2(2-\lambda_i)^2)(\sum_{i=1}^s c_i^2\lambda_i)}{(\sum_{i=1}^s c_i^2(2-\lambda_i)^2)(\sum_{i=1}^s c_i^2)} \\ &= \frac{\sum_{i,j} c_i^2c_j^2(\lambda_i-\lambda_j)(\lambda_j-\lambda_i)(4-\lambda_i-\lambda_j)}{(\sum_{i=1}^s c_i^2(2-\lambda_i)^2)(\sum_{i=1}^s c_i^2)} \\ &= -\frac{\sum_{i,j} c_i^2c_j^2(\lambda_i-\lambda_j)^2(4-\lambda_i-\lambda_j)}{(\sum_{i=1}^s c_i^2(2-\lambda_i)^2)(\sum_{i=1}^s c_i^2)} < 0 \end{aligned} \tag{32}$$

Also, we show the asymptotic property of $R(x)$ in gradient descent,

$$\lim_{t\to\infty} R(x^{(t)}) = \lim_{t\to\infty} \frac{\sum_{i=1}^s c_i^2(2-\lambda_i)^{2t}\lambda_i}{\sum_{i=1}^s c_i^2(2-\lambda_i)^{2t}} = \frac{c_2^2\lambda_2}{c_1^2} \cdot \lim_{t\to\infty} \left(\frac{2-\lambda_2}{2-\lambda_1}\right)^{2t} = 0^+ \tag{33}$$

where $x^{(t)}$ is the t-th new point given by gradient descent. So far, we finish the proof of well-definedness of gradient descent with the $\eta = \text{Tr}(X^\top X)/(2 - \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)})$.

Remark 1. In fact, as stated above, $R(x)$ is invariant to the scaling of x, so we could scale x on its magnitude, i.e., making $\|x\| = c_2 \in \mathbb{R}^+$ as a constraint during the gradient descent iteration, all the properties and results still hold.

Remark 2. In the main text, instead of using a vector x, we use a feature matrix X and define our Reyleigh Quotient by $R(X) = \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)}$. In fact, different feature channels of X could be viewed as independent vector signal $x_i \in \mathbb{R}^m$ and for each channel, the same gradient descent analysis is applied. Therefore, we finish the detailed proof for our formulation in the main text, which is of the following form,

$$\min\ R(X),\ \textit{subject to const.}\ \|X\|_F^2. \tag{34}$$

C Learning Rate η and Neighbor Averaging Weight $w(\eta)$

We show the relation of learning rate η and neighbor averaging weight $w(\eta)$ in this section (the derivation is in terms of the main text, so $\nabla R(x)$ does not have factor 2).

$$x' = x - \eta\nabla R(x) = x - \eta\frac{1}{x^\top x}(\Delta x - R(x)x) = (\frac{R(x)-1}{x^\top x}\cdot\eta + 1)(x + \frac{\eta \cdot D^{-\frac{1}{2}}AD^{-\frac{1}{2}}}{(R(x)-1)\cdot\eta + x^\top x}x) \tag{35}$$

Thus, we have,

$$w(\eta) = \frac{\eta}{(R(x)-1)\cdot\eta + x^\top x} = \frac{1}{R(x) - 1 + \frac{x^\top x}{\eta}}, \tag{36}$$

According to the formulation, $w(\eta)$ is a monotonically increasing function on variable η and is valid when $w(\eta) > 0$. Therefore, when $R(x) \geq 1$, the domain $\eta \in [0, \infty)$ and when $R(x) < 1$ (we know from Eqn. (33) that $R(x) \to 0^+$), the domain of the function is bounded, $\eta \in [0, \frac{x^\top x}{1-R(x)})$.

Remark 3. The choice of $\eta = \text{Tr}(X^\top X)/(2 - \frac{\text{Tr}(X^\top \Delta X)}{\text{Tr}(X^\top X)})$ lies in the valid domain for $\forall x \in \mathbb{R}^m$. Also, in the valid domain, with respect to the change of η, $w(\eta)$ can vary in the range $[0, \infty)$ monotonically.

D Proof of Theorem 1

Proof. Given any non-zero signal $x \in \mathbb{R}^n$ and a symmetric matrix $A \in \mathbb{R}^{n \times n}$ (with non-negative eigenvalues), we factorize them in the eigenspace,

$$A = U\Lambda U^\top = \sum_{i=1}^{s} u_i \lambda_i u_i^\top \quad \text{and} \quad x = \epsilon + \sum_{i=1}^{s} c_i \cdot u_i \tag{37}$$

where S is of rank $s \in \mathbb{N}^+$, $U = [u_i]_{i=1}^s$ and $\Lambda = diag(\lambda_1, \cdots, \lambda_s)$ are eigen matrices. $\{c_i \in \mathbb{R}\}_{i=1}^s$ are coordinates of x in the eigenspace and $\epsilon \in \mathbb{R}$ is a component tangent to the eigenspace. In a k-layer SGC, the effect of graph convolution is the same as applying Laplacian smoothing k times,

$$A^k x = \left(\sum_{i=1}^{s} u_i \lambda_i u_i^\top \right)^k \left(\epsilon + \sum_{i=1}^{s} c_i \cdot u_i \right) = \sum_{i=1}^{s} c_i \lambda_i^k u_i. \tag{38}$$

Suppose λ_1 is the largest eigenvalue and $c_1 \neq 0$. We go with infinite number of layers and then have $\lim_{k \to \infty} S^k x \propto u_1$, which means the output is unrelated to the input features x. □

E Mean-subtraction

Background. In the main text, we start with one of the most popular convolution operator A_{rw} and its largest eigenvector $u_1 = \mathbf{1} \in \mathbb{R}^n$. Let us use a simplified notation $\bar{u_1} = \frac{u_1}{\|u_1\|}$. Given any non-zero $x \in \mathbb{R}^n$, the proposed mean-subtraction has the following form,

$$x_{new} \leftarrow x - x_{mean} = x - \frac{\mathbf{1}\mathbf{1}^\top x}{n} = x - \langle x, \bar{u_1} \rangle \cdot \bar{u_1} \tag{39}$$

There are some facts from spectral graph theory that

- A_{sys} and A_{rw} have the same eigenvalues, $1 = \lambda_1 \geq \cdots \lambda_s > 0$.
- If u is an eigenvector of A_{sys}, i.e., $A_{sys} u = \lambda u$. Then $\tilde{D}^{-\frac{1}{2}} u$ is an eigenvector of A_{rm} with the same eigenvalue, i.e., $A_{rm} \tilde{D}^{-\frac{1}{2}} u = \lambda \tilde{D}^{-\frac{1}{2}} u$.
- The eigenvector associated with the largest eigenvalue $\lambda_1 = 1$ of A_{sys} is $u_1 = \tilde{D}^{\frac{1}{2}}$, while for A_{rw}, it is $A_{rw}\mathbf{1} = \lambda_1 \mathbf{1} = \mathbf{1}$.

Mean-subtraction for A_{sys}. Let first discuss the graph convolution operator A_{sys},

$$A_{sys} = \tilde{D}^{-\frac{1}{2}}(I + A)\tilde{D}^{-\frac{1}{2}} = U\Lambda U^\top = \sum_{i=1}^{s} u_i \lambda_i u_i^\top \tag{40}$$

and the signal x as

$$x = \epsilon + \sum_{i=1}^{s} c_i \cdot u_i \tag{41}$$

Then we apply the Laplacian smoothing k times,

$$\begin{aligned} A_{sys}^k x &= \left(\sum_{i=1}^{s} u_i \lambda_i u_i^\top \right)^k \left(\epsilon + \sum_{i=1}^{s} c_i \cdot u_i \right) \\ &= \left(\sum_{i=1}^{s} u_i \lambda_i^k u_i^\top \right) \left(\epsilon + \sum_{i=1}^{s} c_i \cdot u_i \right) = \sum_{i=1}^{s} c_i \lambda_i^k u_i. \end{aligned} \tag{42}$$

which tells that $\lim_{k \to \infty} A_{sym}^k x \propto u_1 = \tilde{D}^{\frac{1}{2}} \mathbf{1}$. The mean-subtraction trick on A_{sym} is of a factor $\tilde{D}^{\frac{1}{2}}$ (suppose mapping $f(x) = \tilde{D}^{\frac{1}{2}} x$ and inverse mapping $f^{-1}(x) = \tilde{D}^{-\frac{1}{2}} x$),

$$x_{new} \leftarrow f^{-1}(f(x) - f(x_{mean})) = \tilde{D}^{-\frac{1}{2}}(1 - \frac{\mathbf{1}\mathbf{1}^\top}{n})\tilde{D}^{\frac{1}{2}} x = x - \frac{\tilde{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^\top \tilde{D}^{-\frac{1}{2}} x}{n}. \tag{43}$$

Therefore, after one layer of mean-subtraction, the signal x would be,

$$\begin{aligned}
x_{new} &\leftarrow f^{-1}(f(x) - f(x_{mean})) \\
&= \tilde{D}^{-\frac{1}{2}}(1 - \frac{\mathbf{1}\mathbf{1}^\top}{n})\tilde{D}^{\frac{1}{2}}\left(\epsilon + \sum_{i=1}^{s} c_i \cdot u_i\right) \\
&= \tilde{D}^{-\frac{1}{2}}(1 - \frac{\mathbf{1}\mathbf{1}^\top}{n})\tilde{D}^{\frac{1}{2}}\sum_{i=1}^{s} c_i \cdot u_i \\
&= \sum_{i=1}^{s} \tilde{D}^{-\frac{1}{2}}(1 - \frac{\mathbf{1}\mathbf{1}^\top}{n})\tilde{D}^{\frac{1}{2}} c_i \cdot u_i \\
&= \sum_{i=1}^{s} c_i \cdot u_i - \sum_{i=1}^{s} \tilde{D}^{-\frac{1}{2}}\frac{\mathbf{1}\mathbf{1}^\top}{n}\tilde{D}^{\frac{1}{2}} c_i \cdot u_i \\
&= \sum_{i=1}^{s} c_i \cdot u_i - \sum_{i=1}^{1} c_i \cdot u_i = \sum_{i=2}^{s} c_i \cdot u_i
\end{aligned} \tag{44}$$

which eliminate the dominant effect of u_1.

Mean-subtraction for A_{rw}. Then for the graph convolution operator A_{rw}, we could do the similar decomposition,

$$\begin{aligned}
A_{rw} &= \tilde{D}^{-1}(I + A) \\
&= \tilde{D}^{-\frac{1}{2}}\left(\tilde{D}^{-\frac{1}{2}}(I + A)\tilde{D}^{-\frac{1}{2}}\right)\tilde{D}^{\frac{1}{2}} \\
&= \tilde{D}^{-\frac{1}{2}}\left(U\Lambda U^\top\right)\tilde{D}^{\frac{1}{2}} \\
&= \tilde{D}^{-\frac{1}{2}}\left(\sum_{i=1}^{s} u_i\lambda_i u_i^\top\right)\tilde{D}^{\frac{1}{2}}
\end{aligned} \tag{45}$$

and for the signal x into $\{\tilde{D}^{-\frac{1}{2}}u\}$ space as

$$x = \epsilon + \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}}u_i \tag{46}$$

Similar we apply the Laplacian smoothing k times,

$$\begin{aligned}
A_{rw}^k x &= \left(\tilde{D}^{-\frac{1}{2}}\left(\sum_{i=1}^{s} u_i\lambda_i u_i^\top\right)\tilde{D}^{\frac{1}{2}}\right)^k \left(\epsilon + \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}}u_i\right) \\
&= \underbrace{\left(\tilde{D}^{-\frac{1}{2}}\left(\sum_{i=1}^{s} u_i\lambda_i u_i^\top\right)\tilde{D}^{\frac{1}{2}}\right)\cdots\left(\tilde{D}^{-\frac{1}{2}}\left(\sum_{i=1}^{s} u_i\lambda_i u_i^\top\right)\tilde{D}^{\frac{1}{2}}\right)}_{k\ terms}\left(\epsilon + \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}}u_i\right) \\
&= \tilde{D}^{-\frac{1}{2}}\left(\sum_{i=1}^{s} u_i\lambda_i^k u_i^\top\right)\tilde{D}^{\frac{1}{2}}\left(\epsilon + \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}}u_i\right) = \sum_{i=1}^{s} c_i\lambda_i^k\tilde{D}^{-\frac{1}{2}}u_i.
\end{aligned} \tag{47}$$

which tells that $\lim_{k\to\infty} A_{rw}^k x \propto \tilde{D}^{-\frac{1}{2}}u_1 = \mathbf{1}$. The mean-subtraction trick on A_{rw} is

$$x_{new} \leftarrow x - x_{mean} = x - \frac{\mathbf{1}\mathbf{1}^\top x}{n} = x - \langle x, \bar{u_1}\rangle \cdot \bar{u_1} \tag{48}$$

Therefore, after one layer of mean-subtraction, the signal x would be,

$$\begin{aligned}
x_{new} &\leftarrow x - x_{mean} \\
&= (1 - \frac{\mathbf{1}\mathbf{1}^\top}{n}) \left(\epsilon + \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i \right) \\
&= (1 - \frac{\mathbf{1}\mathbf{1}^\top}{n}) \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i \\
&= \sum_{i=1}^{s} (1 - \frac{\mathbf{1}\mathbf{1}^\top}{n}) c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i \\
&= \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i - \sum_{i=1}^{s} \frac{\mathbf{1}\mathbf{1}^\top}{n} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i \\
&= \sum_{i=1}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i - \sum_{i=1}^{1} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i = \sum_{i=2}^{s} c_i \cdot \tilde{D}^{-\frac{1}{2}} u_i
\end{aligned} \tag{49}$$

which eliminate the dominant effect of $\tilde{D}^{-\frac{1}{2}} u_1 = \mathbf{1}$.

Remark 4. So far, we discuss the one layer mean-subtraction for both A_{sys} and A_{rw} and also the powering effect of A_{sys} and A_{rw} on arbitrary signal x (c_1 is non-zero). Although we have show that one layer of mean-subtraction could eliminate the dominant eigenvector (once and for all). However, in the main text, we discuss that in the non-linear deep GCN architecture, which means after the ReLU activation function, the effect of dominant eigenvector may still appear. Therefore, we need mean-subtraction layer after applying activation function and iteratively eliminate u_1 or $\tilde{D}^{-\frac{1}{2}} u_1$. Due to the powering effect, they will finally approximate the Fiedler vector,

$$\lim_{k\to\infty} [A_{rw}^k x]_{mean-subtraction} \propto \tilde{D}^{-\frac{1}{2}} u_2 \quad \text{and} \quad \lim_{k\to\infty} [A_{sys}^k x]_{mean-subtraction} \propto u_2 \tag{50}$$

F *Karate* Demonstration

Mean-subtraction for *Karate*. We use the mean-subtraction trick on *Karate* data. The experiment setting is as follows: we randomly assign 2-dimensional feature vector for each node and apply Laplacian smoothing k times ($k = 0, 5, 20, 100$) with normalized random-walk adjacency operator A_{rw}. For each k, we visualize the feature vector of each node after scaling the dimension by the largest absolute value in that dimension (i.e., f = f / max(abs(f))). From ground truth, each color indicates a class and we manually add them to help with the visualization. It is impressive that with mean-subtraction, nodes are almost well-separated during multi-layer Laplacian smoothing. As is stated in the main text, the reason is that mean-subtraction magnifies the Fiedler vector and achieves a pre-separation effect.

The Cosine Similarity. Suppose the feature matrix after the l-th layer is $X^{(l)} \in \mathbb{R}^{n\times m}$,

$$X^{(l)} = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \\ x_{n1} & & x_{nm} \end{bmatrix} \tag{51}$$

we use $X_i,\ i = 1..n$ to denote the rows of $X^{(l)}$ and use $X^j,\ j = 1..m$ to denote the cols of $X^{(l)}$. $\{X_i\}$ is row vector and $\{X^j\}$ is column vector. The feature-wise smoothing (cosine similarity) is given by the averaging absolute value of the following matrix,

$$\text{avg} \begin{bmatrix} \frac{X^{1\top}X^1}{\|X^1\|\|X^1\|} & \frac{X^{1\top}X^2}{\|X^1\|\|X^2\|} & \cdots & \frac{X^{1\top}X^m}{\|X^1\|\|X^m\|} \\ \frac{X^{2\top}X^1}{\|X^2\|\|X^1\|} & \frac{X^{2\top}X^2}{\|X^2\|\|X^2\|} & \cdots & \frac{X^{2\top}X^m}{\|X^2\|\|X^m\|} \\ \vdots & \vdots & \ddots & \\ \frac{X^{m\top}X^1}{\|X^m\|\|X^1\|} & \frac{X^{m\top}X^2}{\|X^m\|\|X^2\|} & & \frac{X^{m\top}X^m}{\|X^m\|\|X^m\|} \end{bmatrix} \tag{52}$$

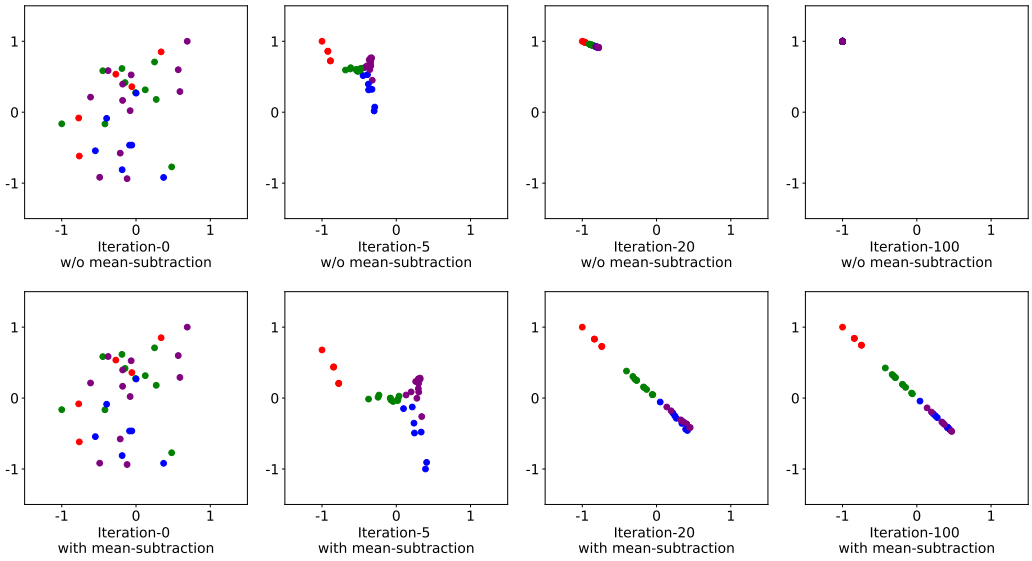

Figure 6: Laplacian Smoothing for *Karate* with or w/o Mean-subtraction

The maximum possible score is 1 if all the entry are either 1 or -1, which means all of they are entirely on the same direction.

Similarly, the node-wise smoothing (cosine similarity) is given by the averaging absolute value of the following matrix,

$$\text{avg}\begin{bmatrix} \frac{X_1X_1^\top}{\|X_1\|\|X_1\|} & \frac{X_1X_2^\top}{\|X_1\|\|X_2\|} & \cdots & \frac{X_1X_n^\top}{\|X_1\|\|X_n\|} \\ \frac{X_2X_1^\top}{\|X_2\|\|X_1\|} & \frac{X_2X_2^\top}{\|X_2\|\|X_2\|} & \cdots & \frac{X_2X_n^\top}{\|X_2\|\|X_n\|} \\ \vdots & \vdots & \ddots & \\ \frac{X_nX_1^\top}{\|X_n\|\|X_1\|} & \frac{X_nX_2^\top}{\|X_n\|\|X_2\|} & & \frac{X_nX_n^\top}{\|X_n\|\|X_n\|} \end{bmatrix} \tag{53}$$

The maximum possible score is 1 if all the entry are either 1 or -1, which means all of they are entirely on the same direction.

G Experimental Details and More

Experimental Setting.

- overfitting in Deep GCNs.

model	trials	layers	epochs	dropout	learning rate	hidden units	Adjacency
GCN with redisual connection	20	1-50	1000	0.5	0.01	16	A_{rw}

- mean-subtraction.

model	trials	layers	epochs	dropout	learning rate	hidden units	Adjacency
GCN with redisual connection	20	64	500	0.5	0.01	16	A_{rw}

- performance vs learning rate η.

Additional Experiments. We train various deep GCNs and deep NN (from 1 layer to 50 layers) on *Cora, Citeseer, Pubmed*. They have the same loss function $\mathcal{L}_0$. We show the training and test curve in Fig. 7, Fig. 8, Fig. 9, separately.

model	trials	layers	epochs	dropout	learning rate	hidden units	Adjacency
GCN with redisual connection	20	2 or 32	1000	0.5	0.005	16	changes

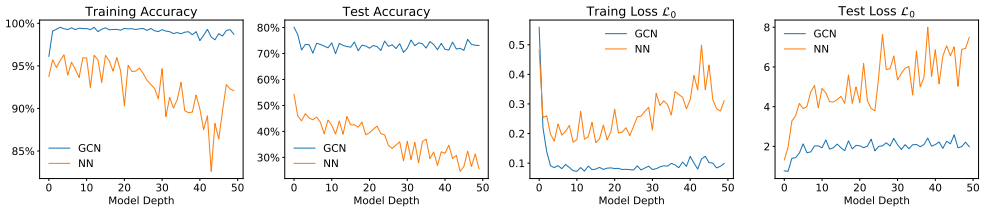

Figure 7: Loss and Accuracy of Training and Test over *Cora*

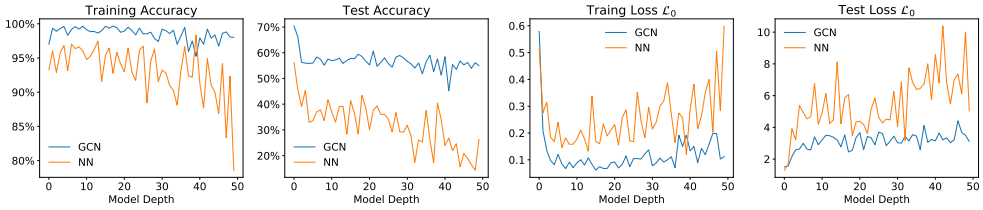

Figure 8: Loss and Accuracy of Training and Test over *Citeseer*

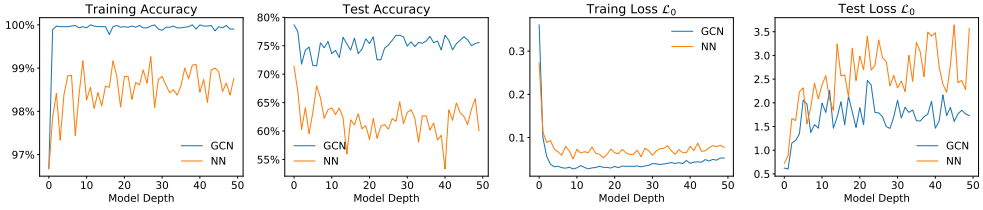

Figure 9: Loss and Accuracy of Training and Test over *PubMed*